Neural learning methods yielding functional invariance*

Vicente Ruiz de Angulo and Carme Torras
Institut de Robòtica i Informàtica Industrial (CSIC-UPC),
Parc Tecnològic de Barcelona - Edifici U,
Llorens i Artigas 4-6, 08028-Barcelona, Spain
ruiz@iri.upc.es, torras@iri.upc.es

Abstract

This paper investigates the functional invariance of neural network learning methods incorporating a complexity reduction mechanism, such as a regularizer. By functional invariance we mean the property of producing functionally equivalent minima as the size of the network grows, when the smoothing parameters are fixed. We study three different principles on which functional invariance can be based, and try to delimit the conditions under which each of them acts. We find out that, surprisingly, some of the most popular neural learning methods, such as weight-decay and input noise addition, exhibit this interesting property.

1 Introduction

This work stems from an observation we made in analyzing the behaviour of a deterministic algorithm to emulate neural learning with random weights [5,7]. We found that, for a fixed variance greater than zero, there is a number of hidden units above which the learned function does not change, or the change is slight and tends to zero as the size the network grows [6]. Here we study the conditions a neural learning algorithm should satisfy in order to lead to the same function, irrespective of network size.

Methods for complexity reduction [1] usually include one parameter (and sometimes more than one) to regulate the simplicity or smoothness imposed on the function implemented by the network. Each method simplifies the network in a way that is supposed to be optimal for the class of functions that is being approximated. Thus, ideally, the optimal level of smoothing should

*A preliminary version of this work was presented at the International Conference on Artificial Neural Networks (ICANN'01), Vienna, August 2001.

be obtained only by manipulating the above-mentioned parameter. Variability caused by other sources must be considered spurious uncertainty. For example, functionally different minima obtained by the same algorithm when departing from different initial points or when running on different architectures are embarrassing for the practitioner, who would desire to be freed from having to optimize the algorithm also along these lines. In particular, the selection of the number of hidden units of the architecture can influence decisively the result and is computationally cumbersome.

This motivates the interest in complexity minimization methods that show dependence only on the explicit complexity parameter and not on the size of the chosen architecture. However, there are no claims about functional invariance for the known methods, although Neal [4] devised a prior such that the complete bayesian procedure using it can be considered functionally invariant (see Section 3.1). In what follows we put forth some theoretical arguments and present some experimental results indicating that functional invariance may be a rather common phenomenon, even in well-known methods used for a long time by the connectionist community.

We also try to delimit the conditions that a complexity reduction method must satisfy in order to yield functional invariance. The paper focuses on the regularization methods for complexity reduction [1] and those that can be made equivalent to them. Regularization consists in adding a penalty function to the error function that regulates the complexity of the implemented network via a multiplicative factor called regularizer coefficient.

2 The phenomenon: learned-function invariance

We shall first outline in common words what is the phenomenon under study, namely learned-function invariance. Two networks are functionally equivalent if they produce the same output for every possible input. This implies that both networks display the same training and generalization error. A complexity reduction method exhibits learned-function invariance if, given a training set and a positive regularizer coefficient, it produces functionally equivalent networks when their size gets large.

We put forward below some rigorous definitions. Let $F(X, W)$ and $G(X, W')$ denote the input-output functions implemented by two feedforward networks having equal number of input and output units, but different number of hidden units, and with weight vectors W and W', respectively.

Definition 1. The *functional distance* between $F(., W)$ and $G(., W')$ is

$$dist(F(., W), G(., W')) = \frac{1}{Vol(\Omega(s))} \int_{\Omega(s)} (F(X, W) - G(X, W'))^2 dX$$

when s tends to infinity, $\Omega(s)$ being a cube of side s in the input space.

Now, let $M(F, \lambda, T)$ be the optimum weight vector obtained with network F by a learning method M involving some complexity reduction regulated by the parameter λ and applied to a training set T. The name "method" is used here to denote an idealized algorithm, usually characterized by the minimization of an objective function, that always finds global optima.

Definition 2. M is a *regularization method* if

$$M(F, \lambda, T) = \text{argmin}_W C(W; T) = \text{argmin}_W (E(W; T) + \lambda R(W)),$$

where $E(W; T)$ is the standard error function, and $R(W) > 0$ is the regularization term[1].

Finally, let $\{F_n\}$ be a family of one hidden-layer architectures differing only in the number n of hidden units.

Definition 3. The learning method M yields *functional invariance* for the network family $\{F_n\}$ if $dist(F_i(., M(F_i, \lambda, T)), F_{i+1}(., M(F_{i+1}, \lambda, T)))$ tends to zero when i tends to infinity for every $\lambda > 0$.

It is necessary to make some remarks about these definitions. First, we only consider global minima of $C(W; T)$ in the definition of a regularization method $M(F, \lambda, T)$, and not local minima, saddle points or other points resulting from a numerical optimization of $C(W; T)$. Second, all the global minima of $C(W; T)$ must be functionally equivalent, or the distance $dist(F_i(., M(F_i, \lambda, T)), F_{i+1}(., M(F_{i+1}, \lambda, T)))$ would not be well defined. Obviously, it is impossible to fulfill this condition for $\lambda = 0$, but not for $\lambda > 0$. This is related to the explicit exclusion of local minima from the functional invariance definition, since global and local minima, having different values of E, produce different outputs for the training patterns, which implies that the two minima cannot be functionally equivalent. Third, it is possible to extend Definition 3 using families of architectures that are not limited to one hidden layer of units (see Section 3.2). The last remark is that the aforementioned definitions do not impose constraints on the training set, the only implicit requirement being that any non-empty training set would bring the functional distance limit to zero.

There is another interpretation of the above definition of learned-function invariance: given a function family $\{F_n\}$ and a training set T, the functional invariance of the algorithm M implies the existence of a target function for each $\lambda > 0$ to which the family tends. We could denote this target or objective function as $\Phi(\lambda, T)$. Overfitting in this context loses its sense. A large number of degrees of freedom does not generate arbitrary features or oscillations. On the contrary, the more hidden units there are, the higher the precision in the approximation of $\Phi(\lambda, T)$. Thus, to ameliorate generalization the practitioner should only concentrate on finding the best smoothing parameters for T, using a number of hidden units according to the computational resources that can be assigned to the task.

[1] R does not usually depend on the training set, although there are exceptions to this (see Section 3.2).

Figure 1 shows two networks trained with the same 20 training points, randomly drawn from the function $0.3x^3+0.3x^2+10/(3(x+3)^2)$ in the interval [-1.5, 1.5], using a deterministic algorithm to emulate learning with random weights [5, 7]: the mean of the weight distribution is adapted to minimize the average error over the distribution. The complexity of the function implemented by that mean is regulated via the variance of the distribution of the weights. Both the four hidden-units (HUs) network and the eight HUs network were trained using the same variance. It can be seen that the distance between the two resulting networks is null, i.e., they are functionally equivalent. Clearly, the weights of the first unit in Figure 1(a) are the same as those of the fifth unit in Figure 1(b). Moreover, the fourth unit in Figure 1(a) shows a direct correspondence with the first unit in Figure 1(b): the weights have pairwise the same magnitude, and since the signs of both the input weights and the output weights are inverted, the two units have the same functionality. It can be concluded that the two networks implement the same approximation of the desired function.

Applying the algorithm to any network with a large number of hidden units, we obtain the same units in different positions and combinations of sign inversions that produce always the same input-output function. Testing the algorithm with other variances produces other configurations that clearly converge to some function in the limit of infinite HUs. However, the closer is the variance to zero, the more difficult is the optimization, as the configurations found become more and more complex, and convergence is attained much more slowly as the number of HUs grows. This is a feature common to all the algorithms that we have explored: it is hard to check functional invariance when the complexity reduction constraints are loose.

3 Conditions for the appearance of invariance

Initially, when we found functional invariance [6] while experimenting with the random weights learning algorithm [5,7], we thought it was a rather unique phenomenon, in the sense that it was particular to the kind of weight configurations that our algorithm created, or at least, that any algorithm exhibiting functional invariance should produce weights sharing the same essential properties. However, a deeper reflection revealed that functional invariance can appear when using different algorithms, and due to very diverse reasons. Up to now, we have identified three types of algorithms corresponding to three principles on which functional invariance can be based.

3.1 Neal's type priors

Regularization methods can be considered under a bayesian perspective by viewing $E(W)$ and $\lambda R(W)$ as the negative log probability (disregarding some constants) of the output distribution of the function being approximated and the

weight prior distribution, respectively. Then, $C(W)$ can be shown to be equivalent to the negative log of the posterior weight probability, and its minimization corresponds to a "maximum a posteriori" procedure.

However, the use of the same weight prior on two different architectures does not imply a direct relationship between the functions they implement. In fact, a prior over the weights in different architectures can induce different priors over the output functions.

Neal [4] devised a prior over the weights that, although inducing also a different prior over functions for each network, converges to a unique one as the number of hidden units tends to infinity.

Convergence of the input-output functions posterior implies convergence of its mode. Thus, a procedure optimizing this posterior may be used to obtain functional invariance. Despite this, a prior over functions in the infinite number of HUs limit is not enough to directly imply the functional invariance of the minimization of $C(W)$. In fact, this minimization finds the most probable weight vector, which *does not correspond* to the most probable input-output function, because there is a Jacobian determinant factor mediating the two probability densities [8], and the minimization of $C(W)$ optimizes the posterior of the weights, not that of the input-output functions.

The true bayesian procedure, however, does not consist simply in finding the mode of the weight posterior, as carried out by the usual regularization method. Instead, it takes into account the complete probability distribution to generate an answer. For example, the bayesian answer to the question "what is the *best* output Y for a given input X?" under a quadratic loss function would be guessing the average of the values for that point of the posterior of the input-output functions, $\int (Y-t)^2 p(t) dt$. This involves an integration over the probability space that cancels out the Jacobian determinant, so that it is the same to integrate over the weight posterior, $\int t\ p(t|W,X) p(W|T) dt$, as to integrate over the input-output function posterior. Thus, this type of answer, considered as the output of the learning algorithm, makes the complete bayesian procedure functionally invariant as we have defined it.

3.2 Regularizers implying a target mean function

Input noise addition during training is equivalent to Tikhonov regularization when the number of patterns of the training set is infinite [1]. Even with finite training sets, it is equivalent to the addition of a penalization term [5, 7], although this is not a classical regularizer because it involves the output training patterns and depends on $E(W;T)$. However, the function invariance property of noise addition is better understood by taking a wider perspective. The minimization of a quadratic $E(W;T)$ can be viewed as an attempt to estimate with the network the mean values taken by the output patterns for a given input. Usually, there are none or very few desired values for each point in the

input space. However, with input noise addition, potentially infinite patterns are available, and the expected output pattern for a given input point is [3]:

$$m(X; p, \{X_i, Y_i\}_{i=1,..n}) = \frac{\sum_{s=1}^{n} Y_s \; p(X - X_s)}{\sum_{k=1}^{n} p(X - X_k)}$$

where $\{X_i, Y_i\}_{i=1,..n}$ are the original (not noisy) training patterns and p is the probability density function of the noise. Note that fixing the type of probability distribution and parametrizing only its variance σ_p as a smoothing parameter, $m(X; p, \{X_i, Y_i\}_{i=1,..n})$ corresponds to the function $\Phi(\lambda, T)$ we talked about in Section 2. We require $p(X)$ to be nonzero for every X in the input space, so that the randomly generated patterns cover it completely. This assures that $m(X)$ is always well-defined, since otherwise the denominator could be zero somewhere. We need $m(X)$ to be well-defined over the entire input space, because the zones of the input space that do not have desired values leave the network free to interpolate arbitrarily. Instead, if a target function is defined over all the input space, any family of networks with enough approximation power has to approximate the same function without degrees of freedom left. Thus, the fundamental condition for noise addition to be a functionally invariant method is the use of infinite domain probability density functions.

This condition could be relaxed by restraining the definition of functional invariance to the domain of real usefulness. This would entail rewriting Definition 1 so as to replace $\Omega(s)$ by Ω_{domain}, where Ω_{domain} is a subspace of the input domain that contains every possible input of the mapping (therefore including also any test set), leading to a weaker definition of functional invariance while retaining all its practical implications. To satisfy this restricted functional invariance, $m(X)$ need to be well-defined only in Ω_{domain}. That is, the ensemble of the training points extended with the noise distribution must cover Ω_{domain}.

An important remark is that the existence and uniqueness of $m(X)$, independently of the architecture used, makes input noise addition functionally invariant in a more general sense, not limited to one hidden-layer architectures. The extension of the definition of functional invariance to multiple hidden-layer architectures is straightforward, the only condition being that the elements of the family of architectures $\{F_n\}$ should be indexable in such a way that, given an arbitrary precision, for any given continuous mapping (from the input to the output space), there exists an index value such that architectures with higher indices can approximate the mapping with that precision.

3.3 Decomposable regularizers

We assume that the networks in this section have hidden units of the weighted-sum type, i.e., given an input X and an incoming weight vector $\mathbf{w}_u^{input}$, the output they produce is $f(\mathbf{w}_u^{input} X)$, where f is a function differentiable at 0.

Let us now talk about regularizers for one-hidden layer networks that are

additively decomposable, $R(W) = \sum_u r(\mathbf{w}_u)$, where $\mathbf{w}_u$ is the vector of all incoming and outcoming weights of hidden unit u. We require also that $r(\mathbf{w}_u)$ has a minimum value of zero attained when $\mathbf{w}_u = 0$[2].

Proposition 1. *Decomposable regularizers exhibit functional invariance if there exists a threshold τ for $r(\cdot)$ such that, when the number of units tends to infinity, the values $r(\mathbf{w}_u) < \tau$ in a global minimum tend to zero, i.e., their associated weights $\mathbf{w}_u$ tend to zero.*

Proof. Suppose we have a network with n hidden units that has been brought to a global minimum of $C(W)$. In the limit of $n = \infty$, if we order the values $r(\mathbf{w}_u)$, the resulting sequence must tend to zero or, otherwise, $R(W) = \infty$ in the minimum. Note that a low $r(\mathbf{w}_u)$, and in particular a low $r(\mathbf{w}_u^{input})$, implies that $\mathbf{w}_u^{input}$ must have a small magnitude and, as a consequence, the hidden units in the sequence can be approximated by $f(0) + f'(0)\ \mathbf{w}_u^{input}\ X$ with increasing accuracy, i.e., tend to be linear. Thus, there exists a finite number N of units whose $r(\mathbf{w}_u)$ is above the threshold and, therefore, are significantly nonlinear. The remaining $n - N$ units contribute globally with a purely linear mapping A to the input of the output layer.

For a network with $n - 1$ hidden units, a configuration with the same N nonlinear units and all but one of the same $n - N$ linear units, reproduces the same function implemented by the n HUs with a difference that tends to zero as n grows. With an appropriate scaling of the linear units, the cost $C(W)$ would be the same because, on the one hand, since the $n - N - 1$ units are linear, the mapping A can be recovered perfectly (therefore keeping $E(W)$) and, on the other hand, since they are on the minimum of $r(\mathbf{w}_u)$, the infinitesimal scaling does not affect the regularization cost, because $\partial r / \partial \mathbf{w}_u = 0$.

This is a global minimum of the $n - 1$ HUs network. If there was a lower minimum with different nonlinear units or a different linear mapping, it would be easy to build a weight configuration in the n HUs having the same $E(W)$ and the same or lower $R(W)$, which would contradict the hypothesis that the original minimum of $C(W)$ for the n HUs network was global.

Thus, the functional distance between two global minima of two architectures differing only in one hidden unit tends to zero when the number of units tends to infinity, which is the definition of functional invariance. □

3.3.1 Learning with random weights

The deterministic algorithm to emulate learning with random weights, which triggered this work, belongs to this category of decomposable regularizers. It relies on the equivalence of weight noise addition with the addition of a decomposable regularizer, which for networks with linear function activations at the

[2] As a matter of fact, it would suffice that the weights from the input layer (excluding the bias unit) to u were zero in the minimum of $r(\mathbf{w}_u)$.

output units takes the form [6,7]:

$$r(\mathbf{w}_u) = a y_u^2 + b \sum_m w_{mu}^2 y_u'^2,$$

where a and b are constants, w_{mu} is the weight from the hidden unit u to the output unit m, and y_u and y'_u are the activation functions of u and its derivative, respectively. It can be observed that this regularizer satisfies the condition of having the minimum at zero only when symmetric activation functions are used.

Experimental results showing the appearance of functional invariance for this algorithm have been presented elsewhere [6] and a very simple example to illustrate the phenomenon has been shown in Figure 1 and discussed at the end of Section 2.

3.3.2 Weight-decay

But the most surprising example of learning algorithm exhibiting functional invariance is the old good weight-decay. In this case, the decomposable regularized is of the form:

$$R(W) = \sum_{u=1}^{n} r(\mathbf{w}_u) = \sum_{u=1}^{n} (w_u^2 + \sum_{i=1}^{p} w_{ui}^2),$$

where n and p are the number of hidden units and the number of inputs, respectively, w_u is the output weight from hidden unit u, and w_{ui} is the weight from input i to hidden unit u.

Careful experiments indicate that it satisfies the condition stated in Proposition 1, namely that there exists a threshold for $r(.)$, such that when the number of units tends to infinity, the weights of all the units u with $r(\mathbf{w}_u)$ under this threshold tend to zero. As an example, see Figure 2, where the weights resulting from training a 4-HU network and a 8-HU network with 25 samples of the function $sin(x_1 + x_2)$ are shown. Linear activation functions were used at the output units. One can see that in Figure 2(a) the first and the fourth units are replicated. In Figure 2(b), the same non-replicated units as in (a) appear, and the two units that were replicated in (a) appear six times in (b) but with smaller weight magnitudes. This is the scaling we were talking about before. When the number of hidden units is very large, the weight magnitudes of the replicated units become practically null, but they keep globally forming the same linear mapping.

We approximate experimentally the functional distance as the mean square distance between the outputs of two networks in a grid of 10,000 points regularly distributed in the input domain. Figure 3 shows the functional distances between architectures with different number of hidden units, minimized for several weight-decay coefficient values α. Since for some α's, some of the networks fell in local minima, in these cases we used the best minimum selected from three different random trials. It is evident that as α grows, all the architectures

tend to produce the same results. But the most interesting observation from this graph is that, for any $\alpha > 0$, the distances between architectures decrease very quickly as the number of hidden units grows, and are indistinguishable from zero above 50 units. Notice that the comparisons involve networks that differ the more in the number of hidden units, the larger the architectures are. The above observation agrees with the expectation of a tendency to closer similarity for larger nets. Of course all the architectures exhibit almost the same generalization error for any positive α, especially those above 50 HUs.

4 Discussion

In this paper we have put forth a definition of functional invariance, which basically states that a learning method is functionally invariant if, when applied to increasingly large networks, the output for every possible input tends to a limit, and have examined what kinds of methods possess this property. We have identified three mechanisms that can originate functional invariance:

- bayesian treatment of neural networks with weight priors that converge to a prior over functions,
- implicit definition of a mean target for the complete input space, and
- additively decomposable regularizers that produce minima with a finite number of nonlinear units in the limit of infinite units.

Examples of the first two types of mechanisms are bayesian learning with Neal's type weight priors, and input noise addition using probability density functions taking non-zero values in their entire domain, respectively. Examples of the third type are the regularizer that emulates learning with random weights and classical weight-decay. There are relations between these mechanisms, but it is difficult to see a unifying principle. For example, the second type can be viewed as defining a prior over input-output functions, as the first type does, namely one that always concentrates the probability on a single function. However, this prior is the same for all networks using the second type of mechanism, while in the first mechanism the prior over functions is approximated only for large networks. In addition, since the target function is completely defined in the second type of mechanism, the distance to that function is directly minimized, whereas in the first type, averaging over the probability distributions is required to guarantee functional invariance.

There are also differences in the type of units that these mechanisms generate. The third type produces only a finite number of nonlinear units in the infinite limit, while the second gives rise to an infinite number of them. Take into account that the implicit target function can be anyone and, therefore, an infinite number of nonlinear units is required to approximate it [2].

It could seem strange that no one (up to our knowledge) had observed the functional invariance of, for example, weight-decay. However, it is not a property that is easily grasped without appropriate experiments. Careful optimizations with different, rather large architectures and a purposive regard at visual representations of weights are required to observe regular patterns in a mesh of eventually many hidden units and see this property with sharpness. Or, alternatively, one should resort to studies of functional distance, which would not be carried out without a good reason.

This does not mean that these results are not of practical relevance; as two different but large networks are brought moderately close to a global minimum, the functional distance between them becomes very small. As for the size of the networks, this functional equivalency begins to be relevant in practice when the network has enough degrees of freedom to approximate reasonably well the implicit target function. This happens with relatively few HUs for the decomposable regularizers when the function is not loosely constrained by the regularizer coefficient. Take into account that the relative importance of the regularizer diminishes with the number of training patterns. Another problem is that, when the number of HUs is close to the number of relevant, nonlinear HUs, to get a good minimization of the criterion function is much harder, because there are many ways to arrive to the global minimum. The probability of getting stuck in local minima, saddle points and other ill-behaved zones seems higher. For that reason and since overfitting is not a problem, we advice to be generous with the number of HUs.

The problem of falling into local minima may be significant in general, but their frequency using weight-decay is apparently not high. For example, in the experiment of Figure 3 comparing the functional distances of several networks with different weight-decay coefficients, among the numerous optimizations required, only three times we got a local minimum in a single trial.

To avoid leaving the reader with the impression that functional invariance is an ubiquituous phenomenon despite its having passed unnoticed until now, let us point out that there are, of course, neural learning methods that do not exhibit this property, such as for example Optimal Brain Damage (see [6] for some experimental evidence).

References

[1] Bishop, C.M. (1995). *Neural networks for pattern recognition.* Oxford University Press.

[2] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2), 251-257.

[3] Koistinen, P. and Holmstrom, L. (1992). Kernel regression and backpropagation training with noise. *Advances in Neural Information Processing*

Systems 4, Morgan-Kauffman.

[4] Neal, R.M. (1996). *Bayesian learning for neural networks*. Springer-Verlag, New York.

[5] Ruiz de Angulo, V. and Torras, C. (1994). Random weights and regularization. *Proc. Intl. Conf. on Artificial Neural Networks (ICANN'94)*, 1456-1459, Sorrento.

[6] Ruiz de Angulo, V. and Torras, C. (2001). Architecture-independent approximation of functions. *Neural Computation*, 13(5), 1119-1135.

[7] Ruiz de Angulo, V. and Torras, C. (2002). A deterministic algorithm that emulates learning with random weights. *Neurocomputing*, 48(1-4), 975-1002.

[8] Wolpert, D.H. (1994). Bayesian backpropagation over I-O function rather than weights. *Advances in Neural Information Processing Systems 6*, Morgan-Kauffman.

Figure captions

Figure 1. The results of training a 4-HU network (a), and an 8-HU network (b) are shown. Each of the blocks in the figure contains all the weights associated to a hidden unit. Weights are represented by squares, white if their value is positive, and black if their value is negative. The size of the square is proportional to the magnitude of the weight. The top weight in the block is the weight from the hidden unit to the output unit. The weights in the bottom row correspond to the weights incoming to the hidden unit.

Figure 2. Results of training a 4-HU network and a 8-HU network with 25 samples of the function $sin(x_1 + x_2)$, using a weight-decay coefficient value of 0.4. The resulting configurations contain replicated units that fill all the spare units available.

Figure 3. Evaluation of the similarity between the functions implemented by architectures with different numbers of hidden units. Values spanning a wide range of the weight-decay coefficient are tested.